Commit ae6c1e9

jsdanielh committed 2 weeks ago · ✓ 8 / 8 · Verified

```
Fix panic in RequestMacroChain with micro block locator

A peer could send a RequestMacroChain with a micro block hash in the
locator list. The handler accepted any on-chain block without checking
if it was a macro block, then called get_macro_blocks() which panics
when given a micro block hash.

Add a macro block type check before accepting a locator hash, skipping
micro blocks and continuing the search.
```

albatross · v1.3.0

1 parent 6ff0800 commit ae6c1e9

2 files changed +224 -4 lines changed

Top

Filter files...

- consensus
  - src/messages
    - handlers.rs
  - tests
    - request_macro_chain.rs

2 files changed +224 -4 lines changed

Search within code

consensus/src/messages/handlers.rs

```
@@ -57,9 +57,16 @@ impl<N: Network> Handle<N, BlockchainProxy> for
RequestMacroChain {
57  57              if let Ok(chain_info) = chain_info
58  58                  && chain_info.on_main_chain
```

```
59  59                  {
60      -               // We found a block, ignore remaining block locator hashes.
61      -               start_block_hash = Some(locator.clone());
62      -               break;
    60  +               // Validate that the locator is a macro block before using it.
    61  +               // This prevents panics when get_macro_blocks() is called with a
        micro block hash.
    62  +               if let Ok(block) = blockchain.get_block(locator, false)
    63  +                   && block.is_macro()
    64  +               {
    65  +                   // We found a macro block on the main chain, use it.
    66  +                   start_block_hash = Some(locator.clone());
    67  +                   break;
    68  +               }
    69  +               // Skip micro blocks and continue searching for a macro block.
63  70              }
64  71          }
65  72          let Some(start_block_hash) = start_block_hash else {
        @@ -75,7 +82,7 @@ impl<N: Network> Handle<N, BlockchainProxy> for
        RequestMacroChain {
75  82                  Direction::Forward,
76  83                  true,
77  84              )
78      -           .unwrap(); // We made sure that start_block_hash is on our chain.
    85  +           .unwrap(); // Safe: We validated that start_block_hash is a macro
        block on our chain.
79  86          let epochs: Vec<_> = election_blocks.iter().map(|block|
        block.hash()).collect();
80  87
81  88          // Add latest checkpoint block if all of the following conditions are
        met:
```

consensus/tests/request_macro_chain.rs

```
    @@ -0,0 +1,213 @@
1   + use std::sync::Arc;
2   +
3   + use nimiq_blockchain::{BlockProducer, Blockchain, BlockchainConfig};
4   + use nimiq_blockchain_interface::AbstractBlockchain;
5   + use nimiq_blockchain_proxy::BlockchainProxy;
```

```
+ use nimiq_consensus::messages::{MacroChainError, RequestMacroChain};
+ use nimiq_database::mdbx::MdbxDatabase;
+ use nimiq_network_interface::request::Handle;
+ use nimiq_network_mock::{MockNetwork, MockPeerId};
+ use nimiq_primitives::{networks::NetworkId, policy::Policy};
+ use nimiq_test_log::test;
+ use nimiq_test_utils::blockchain::{produce_macro_blocks, signing_key,
  voting_key};
+ use nimiq_utils::time::OffsetTime;
+ use parking_lot::RwLock;
+
+ /// Helper to create a test blockchain with some blocks
+ fn create_test_blockchain(num_macro_blocks: usize) -> Arc<RwLock<Blockchain>> {
+     let blockchain = Arc::new(RwLock::new(
+         Blockchain::new(
+             MdbxDatabase::new_volatile(Default::default()).unwrap(),
+             BlockchainConfig::default(),
+             NetworkId::UnitAlbatross,
+             Arc::new(OffsetTime::new()),
+         )
+         .unwrap(),
+     ));
+
+     let producer = BlockProducer::new(signing_key(), voting_key());
+     produce_macro_blocks(&producer, &blockchain, num_macro_blocks);
+
+     blockchain
+ }
+
+ #[test(tokio::test)]
+ async fn test_request_macro_chain_rejects_micro_blocks() {
+     // Create a blockchain with at least one batch (so we have micro blocks)
+     let blockchain = create_test_blockchain(2);
+     let blockchain_proxy = BlockchainProxy::from(&blockchain);
+
+     // Find a micro block hash on the main chain
+     let micro_block_hash = {
+         let blockchain = blockchain.read();
+         let current_block = blockchain.block_number();
+
```

```
45 +         let mut found_hash = None;
46 +         for block_num in 1..=current_block {
47 +             if !Policy::is_macro_block_at(block_num) {
48 +                 if let Ok(block) = blockchain.get_block_at(block_num, false,
   None) {
49 +                     if block.is_micro() {
50 +                         found_hash = Some(block.hash());
51 +                         break;
52 +                     }
53 +                 }
54 +             }
55 +         }
56 +         found_hash.expect("Should have at least one micro block")
57 +     };
58 +
59 +     // Create a request with only the micro block hash as locator
60 +     let request = RequestMacroChain {
61 +         locators: vec![micro_block_hash],
62 +         max_epochs: 10,
63 +     };
64 +
65 +     // Handle the request
66 +     let peer_id = MockPeerId(1);
67 +     let result = <RequestMacroChain as Handle<MockNetwork,
   BlockchainProxy>>::handle(
68 +         &request,
69 +         peer_id,
70 +         &blockchain_proxy,
71 +     );
72 +
73 +     // Should return UnknownLocators error instead of panicking
74 +     assert!(result.is_err());
75 +     assert!(matches!(
76 +         result.unwrap_err(),
77 +         MacroChainError::UnknownLocators
78 +     ));
79 + }
80 +
81 + #[test(tokio::test)]
82 + async fn test_request_macro_chain_accepts_macro_blocks() {
```

```
83  +     // Create a blockchain with multiple macro blocks
84  +     let blockchain = create_test_blockchain(3);
85  +     let blockchain_proxy = BlockchainProxy::from(&blockchain);
86  +
87  +     // Get the first macro block hash from the chain (not the last one)
88  +     let macro_block_hash = {
89  +         let blockchain = blockchain.read();
90  +
91  +         // Find the first macro block (genesis or first checkpoint)
92  +         let mut found_hash = None;
93  +         for block_num in 1..=blockchain.block_number() {
94  +             if Policy::is_macro_block_at(block_num) {
95  +                 if let Ok(block) = blockchain.get_block_at(block_num, false,
    None) {
96  +                     if block.is_macro() {
97  +                         found_hash = Some(block.hash());
98  +                         break; // Use the first one, not the last
99  +                     }
100 +                 }
101 +             }
102 +         }
103 +         found_hash.expect("Should have at least one macro block")
104 +     };
105 +
106 +     // Create a request with the macro block hash as locator
107 +     let request = RequestMacroChain {
108 +         locators: vec![macro_block_hash],
109 +         max_epochs: 10,
110 +     };
111 +
112 +     // Handle the request
113 +     let peer_id = MockPeerId(1);
114 +     let result = <RequestMacroChain as Handle<MockNetwork,
    BlockchainProxy>>::handle(
115 +         &request,
116 +         peer_id,
117 +         &blockchain_proxy,
118 +     );
119 +
120 +     // Should succeed
```

```
121 +     assert!(result.is_ok());
122 +     let response = result.unwrap();
123 +
124 +     // Should return macro blocks after the locator (or be empty if locator is the last block)
125 +     // Since we used the first macro block, there should be more blocks after it
126 +     assert!(!response.epochs.is_empty() || response.checkpoint.is_some());
127 + }
128 +
129 + #[test(tokio::test)]
130 + async fn test_request_macro_chain_skips_micro_blocks() {
131 +     // Create a blockchain with multiple batches
132 +     let blockchain = create_test_blockchain(3);
133 +     let blockchain_proxy = BlockchainProxy::from(&blockchain);
134 +
135 +     // Find both a micro block and a macro block (not the last macro block)
136 +     let (micro_block_hash, macro_block_hash) = {
137 +         let blockchain = blockchain.read();
138 +         let current_block = blockchain.block_number();
139 +
140 +         let mut micro_hash = None;
141 +         let mut macro_hash = None;
142 +
143 +         // Find first micro and first macro block
144 +         for block_num in 1..=current_block {
145 +             if let Ok(block) = blockchain.get_block_at(block_num, false, None) {
146 +                 if block.is_micro() && micro_hash.is_none() {
147 +                     micro_hash = Some(block.hash());
148 +                 } else if block.is_macro() && macro_hash.is_none() {
149 +                     macro_hash = Some(block.hash());
150 +                 }
151 +
152 +                 if micro_hash.is_some() && macro_hash.is_some() {
153 +                     break;
154 +                 }
155 +             }
156 +         }
157 +
```

```
158 +         (
159 +             micro_hash.expect("Should have at least one micro block"),
160 +             macro_hash.expect("Should have at least one macro block"),
161 +         )
162 +     };
163 +
164 +     // Create a request with micro block first, then macro block
165 +     // The handler should skip the micro block and use the macro block
166 +     let request = RequestMacroChain {
167 +         locators: vec![micro_block_hash, macro_block_hash],
168 +         max_epochs: 10,
169 +     };
170 +
171 +     // Handle the request
172 +     let peer_id = MockPeerId(1);
173 +     let result = <RequestMacroChain as Handle<MockNetwork,
      BlockchainProxy>>::handle(
174 +         &request,
175 +         peer_id,
176 +         &blockchain_proxy,
177 +     );
178 +
179 +     // Should succeed by using the macro block
180 +     assert!(result.is_ok());
181 +     let response = result.unwrap();
182 +
183 +     // Should return macro blocks starting from the macro block locator (or
      checkpoint)
184 +     assert!(!response.epochs.is_empty() || response.checkpoint.is_some());
185 + }
186 +
187 + #[test(tokio::test)]
188 + async fn test_request_macro_chain_too_many_locators() {
189 +     let blockchain = create_test_blockchain(2);
190 +     let blockchain_proxy = BlockchainProxy::from(&blockchain);
191 +
192 +     // Create a request with too many locators (> MAX_LOCATORS = 100)
193 +     let locators = vec![Default::default(); 101];
194 +     let request = RequestMacroChain {
195 +         locators,
```

```
196 +          max_epochs: 10,
197 +      };
198 +
199 +      // Handle the request
200 +      let peer_id = MockPeerId(1);
201 +      let result = <RequestMacroChain as Handle<MockNetwork,
    BlockchainProxy>>::handle(
202 +          &request,
203 +          peer_id,
204 +          &blockchain_proxy,
205 +      );
206 +
207 +      // Should return TooManyLocators error
208 +      assert!(result.is_err());
209 +      assert!(matches!(
210 +          result.unwrap_err(),
211 +          MacroChainError::TooManyLocators
212 +      ));
213 + }
```

Comments 0

Please sign in to comment.